# Die Java Sound API

# Java Sound API

Gliederung

- Allgemeines und Grundlagen
- Einführung in die Java Sound API

Gesampelter Sound

↳ Programmieraufgabe

Midi Sound

↳ Programmieraufgabe

# Allgemeines und Grundlagen

JDK 1.1:
- Soundausgabe nur auf Applets möglich
- kein Midi Sound
- Soundausgabe auf AU-Format beschränkt

JDK 1.2:
- WAV, AIFF, MIDI Formate unterstützt

JDK 1.3:
- Soundaufnahme möglich (Midi/Sampled)
- Zusatzgeräte können angesteuert werden

# JDK 1.3/1.4

- Sound API nicht leicht zu bedienen, da:
  - Low-Level-API: schon bei einfachen Effekten viel Programmieraufwand
  - macht wenig Annahmen über standardmäßig verfügbare Hardware
  - Umgang erfordert grundlegendes Verständnis auf diesem Gebiet
- Sound API Basis für alle Arten von Sound in Java
  (z.B. Applikationen, Spiele, Telefon und Konferenz Anwendungen)
- Basis höherer Schnittstellen: Java Media Framework

## Arten von Sound

- Sampled
- MIDI

## 4 Packages

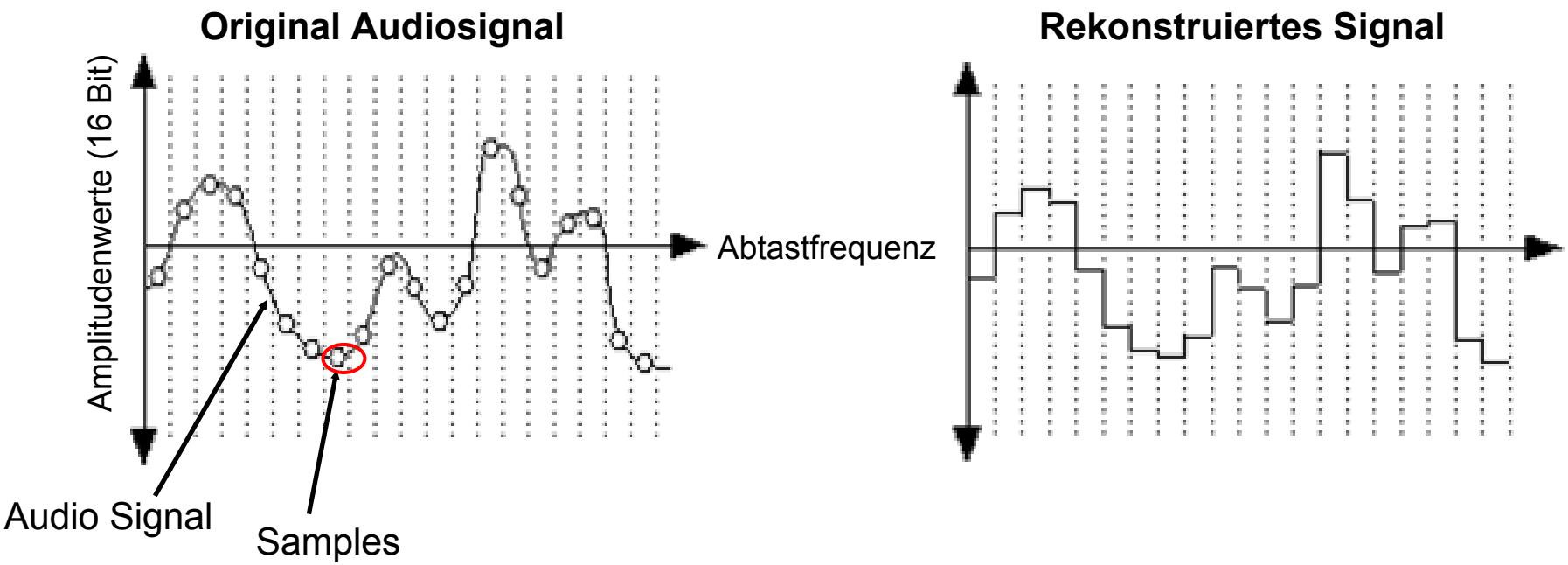
- javax.sound.sampled
- javax.sound.midi

- javax.sound.sampled.spi
- javax.sound.midi.spi

} Service Provider Interface: Für Hersteller

# Gesampelter Sound

## Sampling




- Sampling Rate sollte mindestens doppelt so hoch sein wie die größte aufzuzeichnende Frequenz (Audio CDs: 44100Hz, 16Bit)

# Überblick über das Sampling Package

## Klassen

AudioFileFormat, AudioFileFormat.Type
AudioFormat, AudioFormat.Encoding
AudioInputStream
AudioSystem

DataLine.Info
Line.Info
Mixer.Info
Port.Info

LineEvent, LineEvent.Type
ReverbType
AudioPermission

Control, Control.Type
FloatControl, FloatControl.Type
EnumControl, EnumControl.Type
BooleanControl, BooleanControl.Type
CompoundControl,
CompoundControl.Type

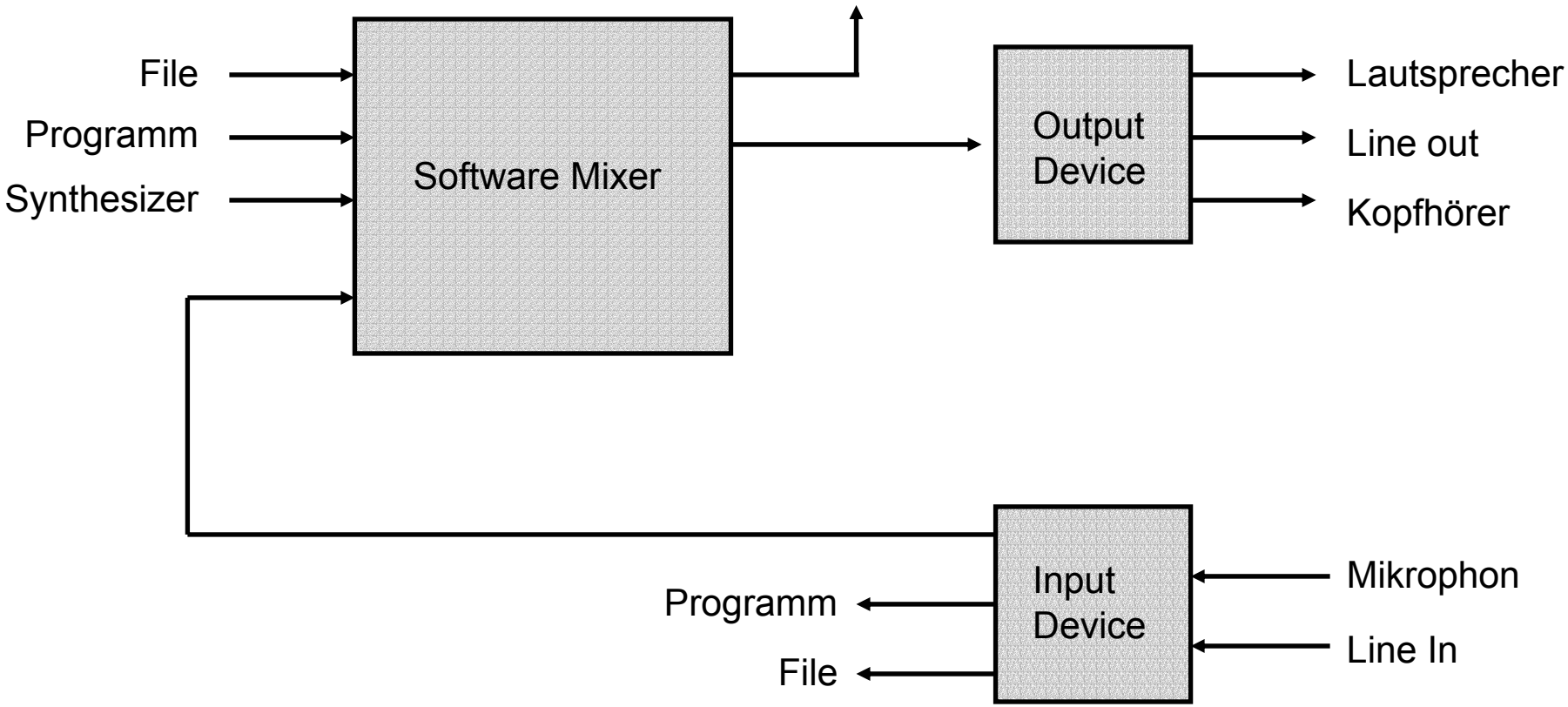
## Interfaces

Clip
DataLine
Mixer
SourceDataLine
TargetDataLine
Port
Line, LineListener

## Ausnahmen

LineUnavailableException
UnsupportedAudioFileException

# Typische Audio Anordnung



## Einführung in Sampled Audio

Zwei Arten für Datentransport von Audiodaten:

- ungepufferter / in-memory Datentransport
- gepufferter / streamed Datentransport

Drei wichtigste Objekte für Umgang mit Audiodaten:

- formatierte Audiodaten
- Mixer
- Line

# Formatierte Audiodaten

## AudioFormat

- Format von gesampelten Sound

Info über:

- Kodierungsverfahren (PCM, a-law, µ-law)
- Anzahl der Kanäle (Mono, Stereo)
- Sampling Rate
- Auflösung der Samples
- Frame Rate
- Größe eines Frames in Bytes
- Byte Order (Big oder little Endian)

## AudioFileFormat

- Format von Dateien, die gesampelten Sound enthalten

Info über:

- Datei Typ (.wav, .au, .aiff)
- Datei Größe in Bytes
- AudioFormat

# Mixer

- Audiogerät mit mehreren Lines
- mixt mehrere Audiodaten zusammen

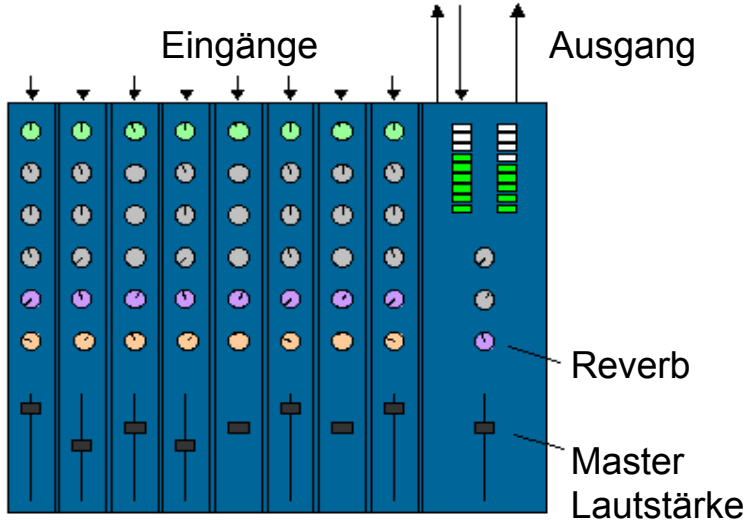
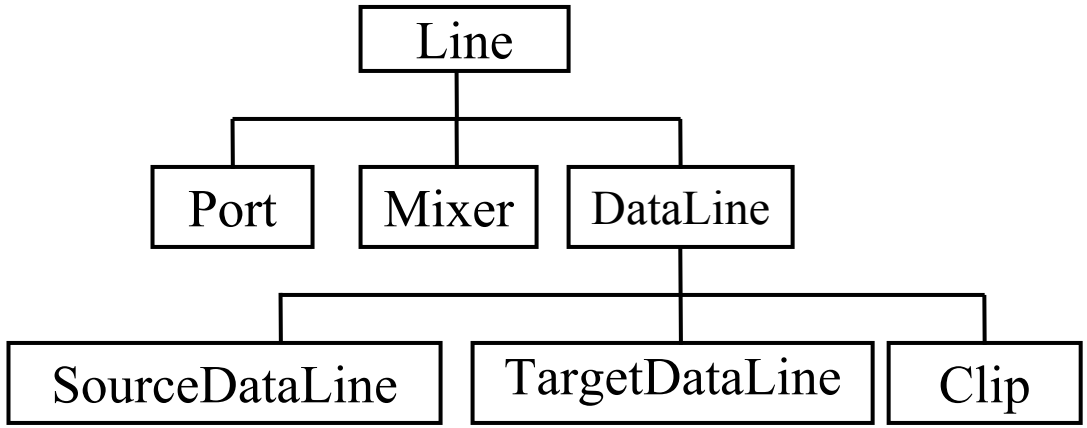



Abb. Physikalischer Mixer

- getLine(Line.info info)
  - ↳ Gibt gewünschte Line aus
- getSourceLineInfo() / getTargetLineInfo()
  - ↳ Gibt entsprechendes Line.Info Objekt aus
- isLineSupported(Line.Info info)
  - ↳ Ob bestimmte Line unterstützt wird
- synchronize (Line[] lines, boolean maintainSync)
  - ↳ Kann mehrere Lines synchronisieren

# Line Interface - Hierachie



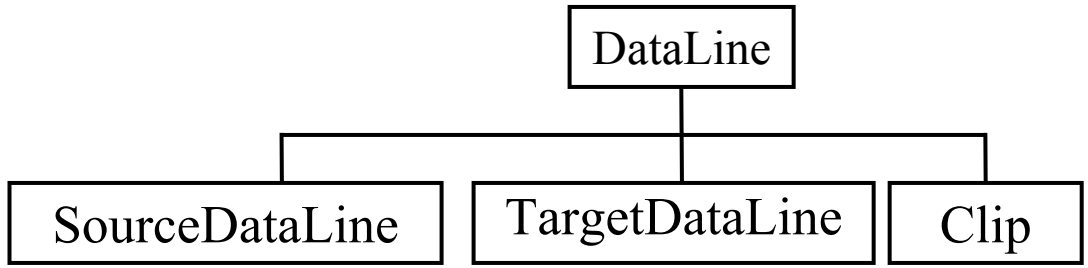

- Weg, auf dem Audiodaten ins oder aus dem System gelangen
- mono oder stereo
- Kontroll-Elemente (gain, pan, reverb)
  - isControlSupported(Control.Type control)
  - getControl(Control.Type control)
- System Ressourcen reservieren / freigeben
  - open() / close()

Port:

↳ Einfache Art einer AudioLine

(Aber Ports sind noch nicht nutzbar)

# DataLine

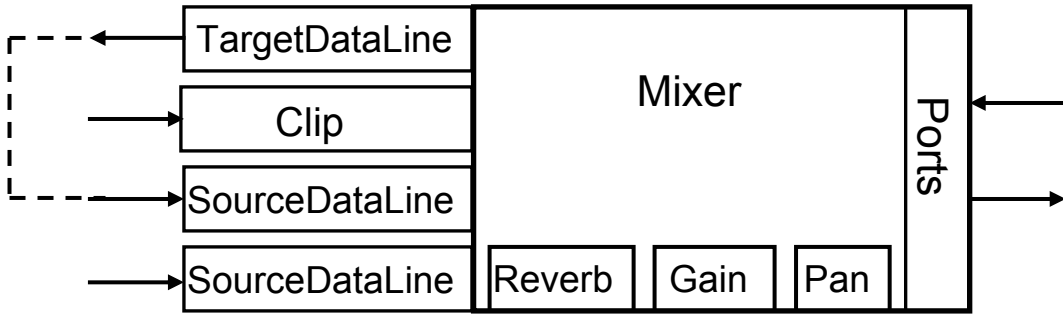



- fügt medien-relevante Funktionen hinzu
- besitzt Puffer, in dem die Daten gespeichert werden können

Zusätzliche Featuers :

- start() / stop() (Abspielen oder Anhalten)
- drain() (blockiert, bis alle Daten aus dem Puffer gelesen worden sind)
- isActive() (ob Line bereits benutzt wird)
- flush() (entfernt alle Daten vom Puffer)

# Unterklassen von DataLine




- Clip:(ungepuffertes Abspielen)
  - ↳ wird vor Weitergabe der Audiodaten komplett in Hauptspeicher geladen
    - loop(int count)

- SourceDataLine (gepuffertes Abspielen eines Datenstroms)
  - ↳ versorgt Mixer Stück für Stück mit Daten
    - write(byte[] b, int off, int len)

- TargetDataLine
  - ↳ empfängt Daten vom Mixer
    - read(byte[] b, int off, int len)

# AudioSystem(1) – Mixer beschaffen

- public static Mixer.Info[] getMixerInfo()
  - ↳ Mixer die installiert sind

    Mixer.Info-Objekt:
    - Name (getName())
    - Version (getVersion())
    - Hersteller (getVendor())
    - Beschreibung (getDescription())
- public static Mixer getMixer(Mixer.Info info)
  - ↳ Beschafft den gewünschten Mixer

```
Mixer.Info[] mixerInfo;
Mixer mixer;

mixerInfo = AudioSystem.getMixerInfo();
Mixer = AudioSystem.getMixer(mixerInfo[0]);
```

# AudioSystem(2) – gewünschte Line beschaffen

von einem Mixer

↳ [Mixer-Objekt].getLine(Line.Info info);

vom AudioSystem

↳ • public static Line getLine(Line.Info info) throws LineUnavailableException

• Line.Info:

- Information über die Klasse der gewünschte Line

Erzeugung des Line.Info-Objekt:

```
Clip clip;
DataLine.Info info = new DataLine.Info(Clip.class, format);
        //format ist ein AudioFormat
```

# AudioSystem(3)

**Zugriff auf Files und Streams:**

- public static AudioInputStream getAudioInputStream(File file)
  (Url url)
  (InputStream stream)

  ↳ beschafft AudioInputStream, mit dem der Inhalt des File ausgelesen werden kann

**Konvertierungen zwischen AudioFormaten:**

- public static boolean isConversionSupported(AudioFormat target Format,
  AudioFormat sourceFormat)

- public static AudioInputStream getAudioInputStream(AudioFormat targetFormat,
  AudioInputStream sourceStream)

# Die Control-Klasse

- Mixer/Line haben Kontrollelemente: Lautstärke, Stereopanorama, Hall

Unterklassen von Control

- FloatControl (Lautstärke)
- BooleanControl (Mute)
- EnumControl (Reverb)
- CompoundControl (Multi Kontroll-Module, Equalizer)

Control Objekt einer Line beschaffen:

- public Control[] getControls()
- public boolean isControlSupported(Control.Type control)
- public Control getControl (Control.Type control)

Wichtigsten Methoden von FloatControl

- public void setValue(float newValue);
- public float getValue();
- public float getMaximum/getMinimum();

Wichtigsten vordefinierte Typen

MASTER_GAIN

PAN

# Abspielen von Audiodaten mit einem Clip

```
File file = new File("Sampled.wav");
AudioInputStream stream =
                AudioSystem.getAudioInputStream(file);
AudioFormat format = stream.getFormat();

DataLine.Info info = new DataLine.Info(Clip.class, format);
                    //Line beschaffen
Clip clip = (Clip) AudioSystem.getLine(info);

clip.open(); // reserviert Ressourcen
clip.start();       //Clip wird abgespielt
...
clip.stop();
clip.close();       // Ressourcen wieder freigeben
```

# Abspielen von Audiodaten mit einer SourceDataLine

```
File file = new File("Sampled.wav");
AudioInputStream stream = AudioSystem.getAudioInputStream(file);
AudioFormat format = stream.getFormat();
byte[] ba = new byte[1024];
int anzahlGelBytes = 0;

DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
SourceDataLine sourceLine
            = (SourceDataLine)AudioSystem.getLine(info);
sourceLine.open();
sourceLine.start();

  while(true){
        anzahlGelBytes = stream.read(ba,0,1024);
        if(anzahlGelBytes == -1)
                break;
                sourceLine.write(ba,0,ba.length);
  }
sourceLine.drain();
sourceLine.stop();
sourceLine.close();
```

# Erfassen von Audiodaten mit einer TargetDataLine

```
AudioFormat neuesformat = new AudioFormat((float)11025.0,16,2,true,false);
byte[] ba = new byte[64];
int anzahlGelBytes = 0;
Boolean aufnahme = true;
ByteArrayOutputStream baOut = new ByteArrayOutputStream();

DataLine.Info info = new DataLine.Info(TargetDataLine.class, neuesformat);
TargetDataLine targetLine
           = (TargetDataLine)AudioSystem.getLine(info);
targetLine.open(neuesFormat);
targetLine.start();

  while(aufnahme){
        anzahlGelBytes = targetLine.read(ba,0,ba.length);
        baOut.write(ba,0,anzahlGelBytes);
        if(anzahlGelBytes == -1)
                break;
  }

targetLine.drain();
targetLine.stop();
targetLine.close();
```

# MIDI – Musical Instrumental Digital Interface

Was ist MIDI?

- beschreibt keine echten Audiodaten
- wie eine Partitur

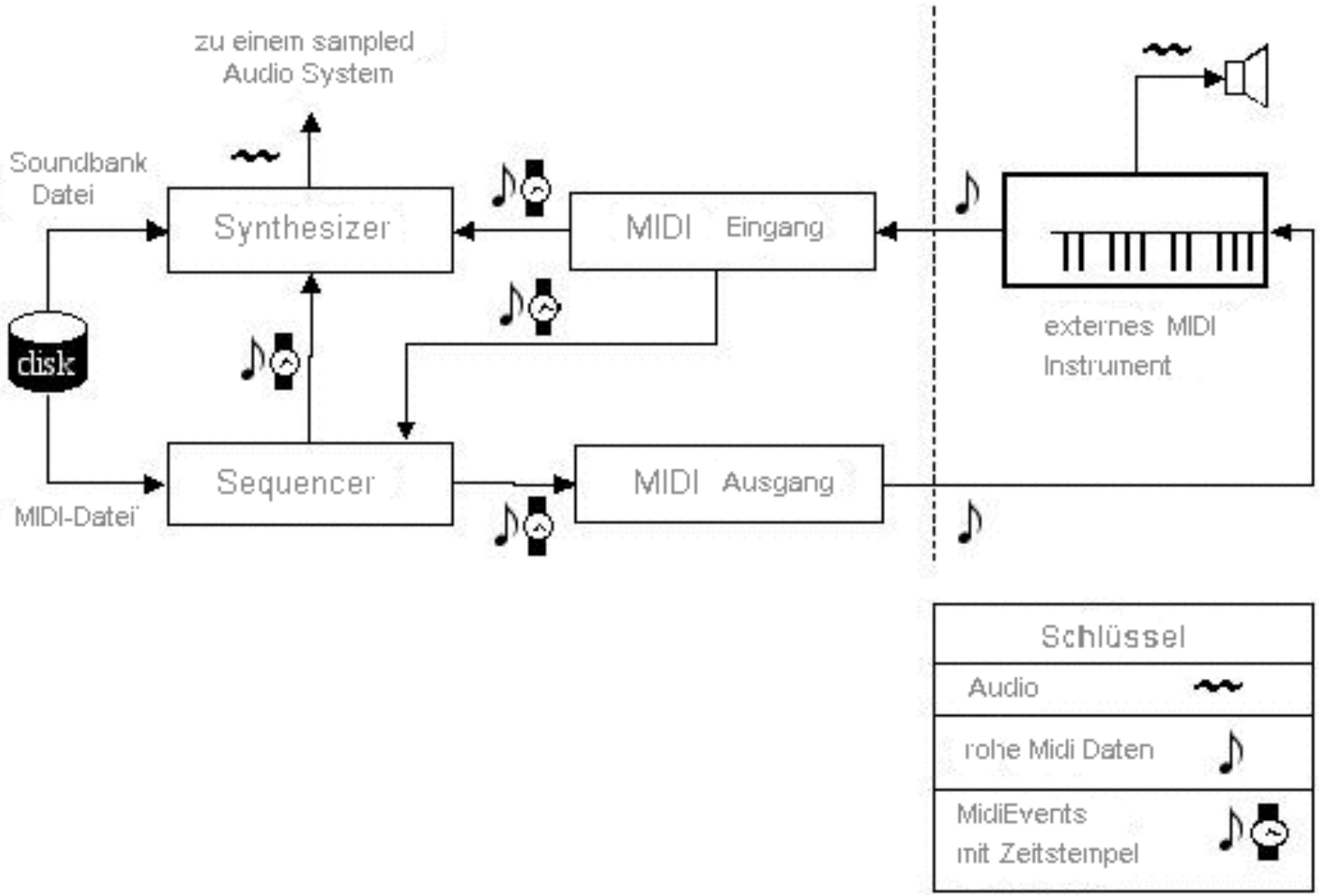
### MIDI Wire Protokoll

- MIDI-Daten byteweise in Echtzeit übertragen

### Standard MIDI Files

- MIDI-Nachrichten mit Zeitstempeln (time stamps)

# Mögliche MIDI Anordnung



# Überblick über das MIDI-Package

## Klassen

MidiSystem

MidiMessage
ShortMessage
MetaMessage
SysexMessage

Sequence
MidiEvent
Track

Instrument
MidiDevice.Info
MidiFileFormat
Patch
Sequencer.SyncMode
SoundbankResource
VoiceStatus

## Interfaces

MidiDevice
Synthesizer
Sequencer
Transmitter
Receiver
MidiChannel
Soundbank
ControllerEventListener
MetaEventListener

## Ausnahmen

InvalidMidiDataException
MidiUnavailableException

# Einführung in MIDI-Sound

Wichtigste Objekte für Umgang mit MIDI-Daten:

- Darstellung von MIDI-Daten
- Transmitter/Receiver
- Sequencer
- Synthesizer

# Darstellung von MIDI-Daten

ShortMessages (SysexMessage, MetaMessage)

- „Rohe“ MIDI-Daten
- z.B. Nachricht, welcher Ton gespielt werden soll
  - setMessage(int command, int channel, data1, data2)

MidiEvents:

- ShortMessage mit Timing-Informationen
  - getMessage()

Tracks

- Track ist eine Sammlung von MidiEvents
  - addMidiEvent(MidiEvent event)
  - removeMidiEvent(MidiEvent event)

Sequences

- Sequence ist eine Sammlung von Tracks
  - createTrack()
  - deleteTrack(Track track)

# MidiDevice-Interface(1) – Transmitter und Receiver

- Weg über den ein Gerät Daten verschickt/empfängt

## Transmitter

- schickt MidiEvents an Receiver
- wird von einem Sequenzer oder Input Port benutzt

  - setReceiver(Receiver receiver)

## Receiver

- empfängt MidiEvents
- verarbeitet MidiEvents zu „rohen“ MIDI-Daten
- wird von einem Synthesizer oder Output Port benutzt

  - send(MidiMessage message, long timeStamp)

# MidiDevice-Interface(2) – Sequencer

- Gerät zum Aufnehmen oder Abspielen von MIDI-Sequenzen
- Transmitter: um MIDI-Events zu schicken
- Receiver: um MIDI-Events zu empfangen

  - getReceiver()
  - getTransmitter()
  - open(), close()
  - start(), stop()
  - startRecording(), stopRecording()
  - recordEnable(Track track, int channel)
  - setSequence(Sequence sequence)
  - getSequence()

# MidiDevice-Interface(3) – Synthesizer

- kontrolliert MidiChannels
- Receiver, der für ihn MidiEvents empfängt
- verarbeitet MidiEvents
- erzeugt Sound, wenn ein MidiChannel eine noteOn – Message erhält

- getReceiver()

- open()/close()

- getChannels()

- getDefaultSoundbank()

- isSoundbankSupported(Soundbank sb)

- getAvailableInstruments()

- loadInstruments(Instrument instrument)

# MidiSystem(1)

Default – Geräte:

- static Sequencer getSequencer()
- static Synthesizer getSynthesizer()
- static Receiver getReceiver()
- static Transmitter getTransmitter()

Installierte Geräte:

- static MidiDevice.Info[] getMidiDeviceInfo()
- MidiDevice.Info:
  - Name (getName())
  - Version (getVersion())
  - Hersteller (getVendor())
  - Beschreibung (getDescription())
- static MidiDevice getMidiDevice(MidiDevice.Info info)

# MidiSystem(2) – Zugriff auf Daten von Standard MIDI-Files

- public static MidiFileFormat getMidiFileFormat(File file),
  (Url url),
  (InputStream stream)

  ↳ Gibt das Dateiformat der MIDI-Datei aus

- public static Sequence getSequence(File file),
  (Url url),
  (InputStream stream)

  ↳ Erhält die MIDI-Sequenz aus einem bestimmtem File

- public static int write(Sequence in, int type, File out)

  ↳ Speichert Sequence in das gewünschte File

# Abspielen von Midi Files

```
Sequencer seq = MidiSystem.getSequencer();
Synthesizer synth = MidiSystem.getSynthesizer();
Transmitter seqTrans = seq.getTransmitter();
Receiver synthRcvr = synthRcvr.getReceiver();
File file = new File(„my.mid“);

      //Verbindung zwischen Transmitter und Receiver schaffen
seqTrans.setReceiver(rcvr);

      //Geräte öffnen
synth.open();
seq.open();
      //Sequence setzen
Sequence sequence = MidiSystem.getSequence(file);
seq.setSequence(sequence);

      //abspielen der Midi Datei
seq.start();
...
seq.stop();
```

# Abspielen von Midi Sound mit dem Synthesizer

```
Synthesizer synth = MidiSystem.getSynthesizer();
int instrument;

   //bestimme verfügbaren Midi-Kanäle
MidiChannel[] channels = synth.getChannels();
   //bestimme Soundbank, die der Synthesizer benutzen soll
Soundbank sb = synth.getDefaultSoundbank();
   //bestimme welche Instrumente die Soundbank zur
   //Verfügung stellt
Instrument[] inst = sb.getInstruments();
... //(Instrument auswählen)
   //Wechsel auf das gewünschte Instrument
channels[0].programChange(instrument);
   //gewünschte Note spielen
channels[0].noteOn(60,93);
...
     // Taste wird nicht mehr gedrückt
channels[0].noteOff(60,93);
```

# Aufnehmen von MidiMessages

```
        //seq schon implementiert
ShortMessage message = new ShortMessage();
        //Geräte öffnen
seq.open();
seqRcvr = seq.getReceiver();
        //leere Sequence erschaffen mit zeitlicher Auflösung
Sequence newSequence = new Sequence(Sequence.PPQ,10);
        //leeren Track erschaffen
Track track = newSequence.createTrack();
seq.setSequence(newSequence);
seq.recordEnable(track,0);
        //Aufnahme starten
seq.startRecording();
        //MidiMessages erschaffen, die man aufzeichnen will z.B.
message.setMessage(ShortMessage.NOTE_ON, 0, 60, 93);
seqRcvr.send(message, -1);
...
        //Aufnahme beenden
seq.stopRecording();
        //File erzeugen, in das man die Daten speichern will
File file = new File(„my.mid“);
MidiSystem.write(newSequence,0,file);
seq.stop();
```

# Literaturangabe


Ausführliche Beschreibung(englisch):

•http://java.sun.com/j2se/1.4.1/docs/guide/sound/programmer_guide/contents.html

• http://java.sun.com/products/java-media/sound/techReference/javasoundfaq.html

•http://www.developer.com/java/other/print.php/1579071

Ausführliche Beschreibung(deutsch):

• http://web.informatik.uni-bonn.de/IV/strelen/Lehre/Veranstaltungen/prak2000/SoundinJava.doc

Sonstiges:

• http://www-cg-hci.informatik.uni-oldenburg.de/~airweb/Seminarphase/DagmarWendt/html/Java.htm

• Krüger, Guido, „Handbuch der Java-Programmierung“, 3.Auflage, Addison Wesley Verlag 2002